```go
package binheap

import (
	"math"
)

type (
	BinaryHeap []Tdata
	Tdata int32
)

const (
	MinData = math.MinInt32
)

func (b *BinaryHeap) Init (data []Tdata) {
	for _, x := range data {
		(*b) = append(*b, x)
	}
	(*b).Heapify()
}

func (b BinaryHeap) pushUp(place int) {
	if place >= len(b) || place <= 0 {
		return
	}
	x := b[place]
	parent:= (place-1)/2
	for  place > 0 && b[parent] < x {
		b[place] = b[parent]
		place = parent
		parent = (place-1)/2
	}
	b[place] = x
}

func (b BinaryHeap) pushDown(place int) {
	if place >= len(b) || place < 0 {
		return
	}
	x := b[place]
	for  {
		if 2*place + 1 >= len(b) { // лист - сыновей нет
			break
		}
		maxson := 2*place + 1  // левый сын
		rson:= maxson + 1
		if rson < len(b) && b[rson] > b[maxson] { // правый сын больше левого
		maxson = rson
		}
		if b[maxson] <= x {
			break
		}
		b[place] = b[maxson]
		place = maxson
	}
	b[place] = x
}

func (b *BinaryHeap) Add(value Tdata) {
	(*b) = append(*b, value)
	(*b).pushUp(len(*b)-1)
}

func (b BinaryHeap) GetMax() Tdata {
	if len(b) > 0 {
		return b[0]
	} else {
		return MinData
	}
}

func (b BinaryHeap) Heapify() {
	for k:= (len(b) / 2) - 1; k >= 0; k-- {
		b.pushDown(k)
	}
}
```

```
78
79  func (b *BinaryHeap) ExtractMax() Tdata {
80        if len(*b) > 0 {
81              max:= (*b)[0]
82              (*b)[0] = (*b)[len(*b)-1]
83              *b = (*b)[:len(*b)-1]
84              (*b).pushDown(0)
85              return max
86        } else {
87              return MinData
88        }
89  }
90
91  func (b *BinaryHeap) Delete(place int) {
92        if place >= len(*b) {
93              return
94        }
95        x:= (*b)[len(*b)-1]
96        *b = (*b)[:len(*b)-1]
97        (*b).Change(place, x)
98  }
99
100 func (b BinaryHeap) Change(place int, value Tdata) {
101       b[place] = value
102       if place > 0 && value > b[(place - 1) / 2] {
103             b.pushUp(place)
104       } else {
105             b.pushDown(place)
106       }
107 }
108
109 type (
110       Lmnt struct {
111             Index int
112             Value Tdata
113       }
114       LocatorBinaryHeap struct {
115             heap []Lmnt
116             locator []int
117       }
118 )
119
120 func (b *LocatorBinaryHeap) Init (data []Tdata) {
121       for i, x := range data {
122             (*b).heap = append((*b).heap, Lmnt{i, x})
123             (*b).locator = append((*b).locator, i)
124       }
125       (*b).Heapify()
126 }
127
128 func (b LocatorBinaryHeap) pushUp(place int) {
129 // поднять элемент, стоящий в b.heap на месте #place
130       if place >= len(b.heap) || place <= 0 {
131             return
132       }
133       t := b.heap[place]
134       parent:= (place-1)/2
135       for  place > 0 && b.heap[parent].Value < t.Value {
136             b.heap[place] = b.heap[parent]
137             b.locator[b.heap[parent].Index] = place
138             place = parent
139             parent = (place-1)/2
140       }
141       b.heap[place] = t
142       b.locator[t.Index] = place
143 }
144
145 func (b LocatorBinaryHeap) pushDown(place int) {
146 // спустить элемент, стоящий в b.heap на месте #place
147       if place >= len(b.heap) || place < 0 {
148             return
149       }
150       t := b.heap[place]
151       for  {
152             if 2*place + 1 >= len(b.heap) { // лист - сыновей нет
153                   break
154             }
```

```
155               maxson := 2*place + 1  // левый сын
156               rson:= maxson + 1
157               if rson < len(b.heap) && b.heap[rson].Value > b.heap[maxson].Value {
158                     // правый сын больше левого
159                     maxson = rson
160               }
161               if b.heap[maxson].Value <= t.Value {
162                     break
163               }
164               b.heap[place] = b.heap[maxson]
165               b.locator[b.heap[maxson].Index] = place
166               place = maxson
167         }
168         b.heap[place] = t
169         b.locator[t.Index] = place
170   }
171
172   func (b LocatorBinaryHeap) Heapify() {
173         for k:= (len(b.heap) / 2) - 1; k >= 0; k-- {
174               b.pushDown(k)
175         }
176   }
177
178   func (b *LocatorBinaryHeap) Add(t Lmnt) {
179         if t.Index < 0 || t.Index >= len((*b).locator) || (*b).locator[t.Index] >= 0 {
180         // Ошибка - кривой индекс или добавляем элемент, который сейчас уже в куче
181               return
182         }
183         (*b).heap = append((*b).heap, t)
184         (*b).locator[t.Index] = len((*b).heap)-1
185         (*b).pushUp(len((*b).heap)-1)
186   }
187
188   func (b LocatorBinaryHeap) GetMax() Lmnt {
189         if len(b.heap) > 0 {
190               return b.heap[0]
191         } else {
192               return Lmnt{-1, MinData}
193         }
194   }
195
196   func (b *LocatorBinaryHeap) ExtractMax() Lmnt {
197         if len((*b).heap) > 0 {
198               max:= (*b).heap[0]
199               (*b).locator[(*b).heap[0].Index] = -1
200               (*b).heap[0] = (*b).heap[len((*b).heap)-1]
201               (*b).heap = (*b).heap[:len((*b).heap)-1]
202               (*b).pushDown(0)
203               return max
204         } else {
205               return Lmnt{-1, MinData}
206         }
207   }
208
209   func (b *LocatorBinaryHeap) Delete(index int) {
210         if index < 0 || index >= len((*b).locator) || (*b).locator[index] == -1 {
211         // Ошибка - кривой индекс или удаляем из кучи элемент,который сейчас не в куче
212               return
213         }
214         place:= (*b).locator[index]
215         if place >= len((*b).heap) {
216               return
217         }
218         (*b).locator[index] = -1
219         t:= (*b).heap[len((*b).heap)-1]
220         (*b).heap[place] = t
221         (*b).locator[t.Index] = place
222         (*b).heap = (*b).heap[:len((*b).heap)-1]
223         (*b).Change(t)
224   }
225
```

```
226  func (b LocatorBinaryHeap) Change(t Lmnt) {
227        if t.Index < 0 || t.Index >= len(b.locator) || b.locator[t.Index] == -1 {
228        // Ошибка - кривой индекс или изменяем элемент, который сейчас не в куче
229              return
230        }
231        place:= b.locator[t.Index]
232        x:= b.heap[place].Value
233        b.heap[place] = t
234        if t.Value > x {
235              b.pushUp(place)
236        } else {
237              b.pushDown(place)
238        }
239  }
```

binheap.go

```
1   package main
2   import (
3         "fmt"
4         "binheap"
5   )
6
7   func HeapSort(a []binheap.Tdata) {
8         var bheap binheap.BinaryHeap
9         bheap.Init(a)
10        for k:= len(a) - 1; k >= 0; k-- {
11              a[k] = bheap.ExtractMax()
12        }
13  }
14
15  func main() {
16        a:= []binheap.Tdata{2,5,7,2,4,9,1,6}
17        fmt.Println(a)    // [2 5 7 2 4 9 1 6]
18        HeapSort(a)
19        fmt.Println(a)    // [1 2 2 4 5 6 7 9]
20  }
```

heapsort.go

```
1   package main
2   import (
3         "fmt"
4         "binheap"
5   )
6
7   func HeapSort(a []binheap.Tdata) []binheap.Tdata{
8         var ( b binheap.LocatorBinaryHeap
9               t binheap.Lmnt
10        )
11        b.Init(a)
12        result:= make([]binheap.Tdata, len(a), len(a))
13        for k:= len(a) - 1; k >= 0; k-- {
14              t = b.ExtractMax()
15              result[k] = t.Value
16        }
17        return result
18  }
19
20  func main() {
21        a:= []binheap.Tdata{2,5,7,2,4,9,1,6}
22        fmt.Println(a)                  // [2 5 7 2 4 9 1 6]
23        fmt.Println(HeapSort(a))        // [1 2 2 4 5 6 7 9]
24        a = []binheap.Tdata{3,7,2}
25        var b binheap.LocatorBinaryHeap
26        b.Init(a)
27        fmt.Println(b)    // {[{1 7} {0 3} {2 2}] [1 0 2]}
28        b.Delete(1)
29        fmt.Println(b)    // {[{0 3} {2 2}] [0 -1 1]}
30        b.Add(binheap.Lmnt{1,17})
31        fmt.Println(b)    // {[{1 17} {2 2} {0 3}] [2 0 1]}
32        b.Add(binheap.Lmnt{1,7})
33        fmt.Println(b)    // {[{1 17} {2 2} {0 3}] [2 0 1]}
34  }
```

locsort.go